Pathway analysis of IMC

Skrypnyuk, Nataliya; Nielson, Flemming; Pilegaard, Henrik

Published in:
21st Nordic Workshop on Programming Theory

Publication date:
2009

Pathway Analysis for IMC

Nataliya Skrypnyuk Flemming Nielson
Henrik Pilegaard
DTU Informatics, Technical University of Denmark
{nsk|nielson|hepi}@imm.dtu.dk

Extended abstract

Abstract

We present the ongoing work on the pathway analysis of a stochastic calculus. Firstly we present a particular stochastic calculus that we have chosen for our modeling - the Interactive Markov Chains calculus, IMC for short. After that we specify a few restrictions that we have introduced into the syntax of IMC in order to make our analysis feasible. Finally we describe the analysis itself together with several theoretical results that we have proved for it.

The IMC calculus has been introduced by Holger Hermanns in the 90's as an orthogonal extension of Continuous Time Markov Chains (CTMC) and a process algebra (see [BH01]). We have adopted the syntax of IMC with a few minor changes (see Table 1). Small Latin letters denote actions that IMC processes are able to execute, capital Latin letters indicate process identifiers and Greek letters correspond to positive real numbers standing for delays during which IMC processes are stuck, i.e. do nothing. Delay durations are not fixed but are exponentially distributed with rates equal to the corresponding numbers. If the rate of some delay is equal to λ, then it's average duration will be $1/\lambda$ and it's variation - $1/\lambda^2$. One difference with respect to [BH01] is that all summands in the sum construct are "guarded" either by actions or delay rates (see rule (2) in the Table 1). This is why we use the name *guarded* IMC or IMC$^{\mathrm{G}}$. Another difference is that all actions and delay rates are decorated with labels from a predefined label set. Labels do not have any semantic meaning but are useful for our analysis.

P	$::=$	X	$\mid$	(1)
		$\sum_{i \in I_1} \mathsf{a}_{\mathsf{i}}^{\ell_i}.P + \sum_{j \in I_2} (\lambda_{\mathsf{j}}^{\ell_j}).P$	$\mid$	(2)
		$\mathsf{hide}\ A\ \mathsf{in}\ P$	$\mid$	(3)
		$P \parallel A \parallel P$	$\mid$	(4)
		$\underline{X := P}$		(5)

Table 1: Syntax of IMC$^{\mathrm{G}}$.

The operational semantics of IMC is defined in terms of a Labeled Transition System (LTS) in the usual way. The novelty of IMC is that delays are dealt with much in the same way as actions. This is justified by the memoryless property of exponential distributions. Namely if two exponentially distributed delays are being executed in parallel and one of them "fires" first, this does not influence the residual duration of the second delay: it will still be exponentially distributed with the same rate. We have enhanced the semantics of IMC in that transitions in LTS are decorated by the multisets of labels of actions or delays being executed. We use this for proving the correctness of our analysis.

The choice (+ operator) between two delays is made according to the *race condition*: the probability that one of two delays "fires" before the other one is equal to it's rate divided by the joint rate of two delays. The time until the first of two delays has "fired"

is exponentially distributed with the rate equal to the sum of two rates. These facts follow from the properties of exponential distributions as well.

It is possible to represent each CTMC as an IMC system and similarly any process algebra expression can be rewritten into an IMC expression. Process algebra part of IMC is defined in the style of Communicating Sequential Processes (CSP) calculus (see [Hoa85]). In particular the synchronization model of CSP is being used. This means that the number of actions being executed synchronously can be any number between 2 and infinity.

The choice between a delay and an internal action (denoted by τ) is made in favor of internal action. This is justified by the notion of the *maximal progress*: as actions (in the approximation) do not take any time to be completed, the choice should also be made in favor of an action according to the race condition. The choice between two actions and between a delay and a non-internal action is made non-deterministically. The last case differs from the case of an internal action because non-internal actions can be blocked by the environment. We should therefore take into account the possibility that the delay will "fire" before the non-internal action will become enabled.

In order to give guarantees for our analysis we need to exclude several kinds of IMC processes with "inconvenient" behavior. In particular we are excluding processes with not well-defined next transactions (for example, $\underline{X := X}$), with an ability to "grow" (like the process $\underline{X := \mathsf{a}^{\ell_1}.X \parallel \{\mathsf{a}\} \parallel \mathsf{a}^{\ell_2}.X}$, which doubles the number of synchronizing subprocesses after each step in the semantics) and processes where there is a choice if some action will be hidden or not (like in the process $\underline{X := \mathsf{a}^{\ell_1}.X + \mathsf{hide}\ \{\mathsf{a}\}\ \mathsf{in}\ \mathsf{a}^{\ell_2}.X}$). In particular the exclusion of "growing" processes guarantees that LTSs of IMC$^{\mathrm{G}}$ processes are always finite.

We have devised a set of inductive well-formedness rules (see Table 2). The operator **fn** returns the set of non-hidden actions and the operator **fpi** returns the set of non-defined IMC$^{\mathrm{G}}$ process identifiers. We name an IMC$^{\mathrm{G}}$ expression well-formed if it can be derived according to these rules parametrized by the set of non-hidden actions of the expressions in question. We have proved that well-formedness is preserved under any number of semantic transitions. We have also shown that all the behavior mentioned above is excluded for well-formed IMC$^{\mathrm{G}}$.

$$\frac{\vdash_S false}{\vdash_S X} \tag{1}$$

$$\frac{\bigwedge_{i \in I_1} (\vdash_S P_i) \qquad \bigwedge_{j \in I_2} (\vdash_S P_j)}{\vdash_S \sum_{i \in I_1} \mathsf{a}_\mathsf{i}^{\ell_i}.P_i + \sum_{j \in I_2} (\lambda_\mathsf{j}^{\ell_j}).P_j + \sum_{i \in J_1} \mathsf{a}_\mathsf{i}^{\ell_i}.X_i + \sum_{j \in J_2} (\lambda_\mathsf{j}^{\ell_j}).X_j} \tag{2}$$

$$\frac{\vdash_S P}{\vdash_S \mathsf{hide}\ A\ \mathsf{in}\ P}\ if \quad A \cap S = \emptyset \tag{3}$$

$$\frac{\vdash_S P_1 \qquad \vdash_S P_2}{\vdash_S P_1 \parallel A \parallel P_2}\ if \quad (\mathbf{fpi}(P_1) = \emptyset) \wedge (\mathbf{fpi}(P_2) = \emptyset) \tag{4}$$

$$\frac{\vdash_{S \cup \mathbf{fn}(P)} P}{\vdash_S \underline{X := P}} \tag{5}$$

Table 2: Well-formedness rules for IMC$^{\mathrm{G}}$ expressions.

We have devised the pathway analysis for IMC$^{\mathrm{G}}$. It is based on the data flow analysis for CCS calculus (see [NN07]) and the pathway analysis for BioAmbients (see [Pil07]). The main contribution in the previous work was that transfer functions of data flow analysis have been applied to the calculi domain. The transfer functions have been computed on the expressions from a particular calculus. The result of the analysis was the construction of the DFA, with each state of the automaton corresponding to one or more states of the underlying LTS semantics.

The reason for doing data flow or similar to it pathway analysis instead of analyzing the underlying LTS semantics directly is to fight the state space explosion problem and even to

be able to deal with the infinite state space. Another potential benefits are more efficient checks of bisimilarity properties, better control over merging of states of the constructed DFA and eventually more efficient model checking of some other classes of properties of the calculi expressions. We can also deal more efficiently with the composition of several systems because we can conduct composition directly on the analysis results for subsystems.

In our own work we have extended the *expose*, *generate* and *kill* operators, already introduced in the previous work, to the IMC^{G} calculus. These operators are defined inductively on IMC^{G} expressions and they return multisets of labeled actions and delay rates. The *expose* operator $\mathcal{E}$ returns labeled actions and delays which may get executed during the next step. The *generate* operator $\mathcal{G}$ returns labeled actions and delays which may become exposed (i.e. available for execution) in one step, after the corresponding labeled action or delay has been executed. The *kill* operator $\mathcal{K}$ returns labeled actions and delays which must cease to be available for execution after the corresponding labeled action or delay has been executed. We have added a new operator, the so-called *chains* operator $\mathfrak{T}$, which returns the set of multisets of labeled actions which have to be executed simultaneously. This has to do with the fact that in the IMC^{G} calculus the number of synchronizing actions is not fixed. The *chains* operator for delays is not necessary because delays do not synchronize in IMC^{G}.

We have established several results for our analysis. We have proved that the analysis is safe for well-formed and "process identifiers closed" IMC^{G} expressions. For such expressions we can compute all executable actions and delays after any number of steps in their semantics. If moreover the IMC^{G} expression in question is uniquely labeled (all labels are different), then the pathway analysis is also precise, i.e. all computed executable actions and delays will in fact be executable in the semantics of the IMC^{G} expression in question.

Theorem 1 *If* $E \xrightarrow{*} E'$ *and* E *well-formed and process identifier closed* IMC^{G} *expression, then from* $E' \xrightarrow[C]{\beta} E''$, β *action or delay rate,* C *multiset of labels, follows* $\mathcal{E}[\![E']\!] + \sum_{\ell \in C} [\mathcal{G}_{\{\}}[\![E]\!]]\ell - \sum_{\ell \in C} [\mathcal{K}[\![E]\!]]\ell = \mathcal{E}[\![E'']\!]$.

Theorem 2 *If* E *is well-formed, process identifier closed and uniquely labeled* IMC^{G} *expression, then in case* $E \xrightarrow{*} E'$ *and some transition is predicted for* E' *by the pathway analysis, this transition can also be derived in the semantics.*

As future work, in order to reduce the state space further, we plan to incorporate techniques for abstracting delays.

References

[BH01] Ed Brinksma and Holger Hermanns. Process Algebra and Markov Chains. In *FMPA 2000: EuroSummerschool on Formal Methods and Performance Analysis, volume 2090 of Lecture Notes in Computer Science*, pages 183–231. Springer-Verlag, 2001.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[NN07] H. R. Nielson and F. Nielson. Data Flow Analysis for CCS. In Program Analysis, Theory Compilation, and Practice, editors, *Conference*, volume 4444 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[Pil07] Henrik Pilegaard. *Language Based Techniques for Systems Biology.* PhD thesis, Technical University of Denmark, Department of Informatics and Mathematical Modeling, Language-Based Technology, 2007.